



Entendiendo el problema



Datos semiestructurados



UDFs



Caso práctico

# Análisis de datos utilizando datos semiestructurados

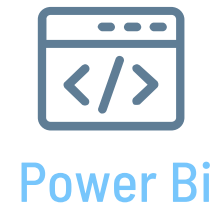
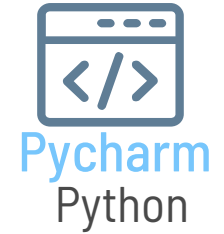
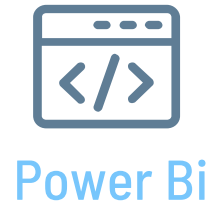
# Entendiendo el problema



Reportes estáticos



Reportes dinámicos



# Caso práctico

## Reducción de dimensión del dato



Datos: Ordenes e-commerce de tienda online OLIST (Brasil)

ORDER_ID	CUSTOMER_ID	FECHA_HORA_OPERACION	TIPO_OPERACION	FECHA_ESTIMADA_ENVIO
00125cb692d04887809806618a2a145f	8afb90a97ee661103014329b1bcea1a2	07/04/2017 15:32	Enviada	20/04/2017 0:00
00125cb692d04887809806618a2a145f	8afb90a97ee661103014329b1bcea1a2	23/03/2017 12:21	Compra	20/04/2017 0:00
00125cb692d04887809806618a2a145f	8afb90a97ee661103014329b1bcea1a2	23/03/2017 13:05	Aprobación de pago	20/04/2017 0:00
00125cb692d04887809806618a2a145f	8afb90a97ee661103014329b1bcea1a2	27/03/2017 8:58	En proceso	20/04/2017 0:00



ORDER_ID	CUSTOMER_ID	FECHA_ESTIMADA_ENVIO	EVENTO
00125cb692d04887809806618a2a145f	8afb90a97ee661103014329b1bcea1a2	20/04/2017 0:00	?



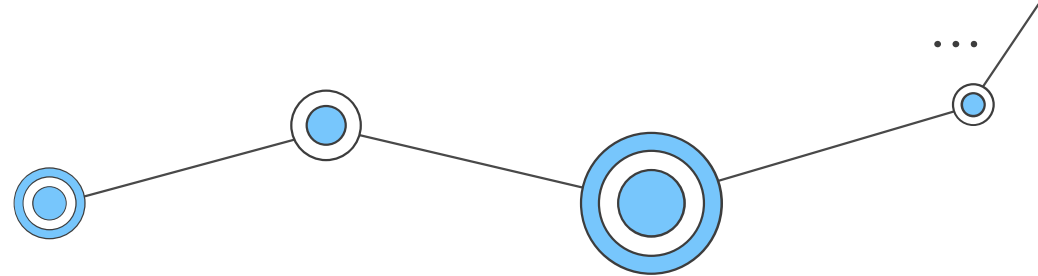
# Datos semi-estructurados

Los datos semiestructurados no tienen un esquema definido. No encajan en un formato de tablas/filas/columnas, sino que se organizan mediante etiquetas que permiten agruparlos y crear jerarquías.

---

## Tipos:

- Variant
- Array (lista)
- Object (diccionario)



# Arrays (listas)

En una lista podemos almacenar dato de una forma ordenada. Ya que sus elementos mantienen una posición.

Ejemplo:

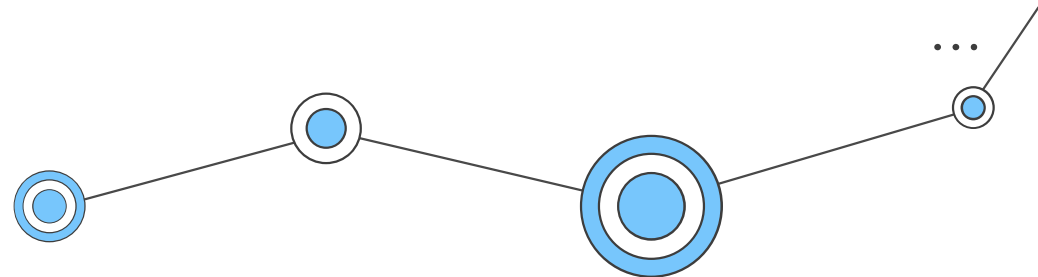
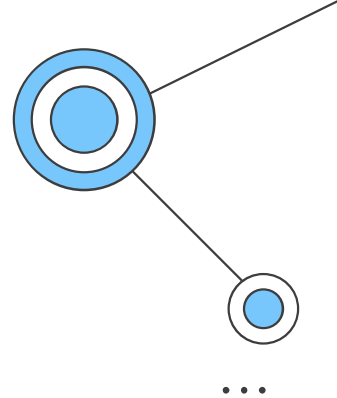
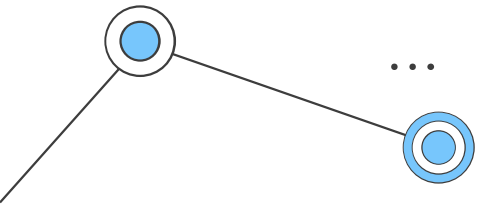
- Crear una lista del tipo\_operación ordenada por fecha\_hora\_operación ascendentemente.

FECHA_HORA_OPERACION	TIPO_OPERACION
07/04/2017 15:32	Enviada
23/03/2017 12:21	Compra
23/03/2017 13:05	Aprobación de pago
27/03/2017 8:58	En proceso



Array

['Compra', 'Aprobación de pago', 'En proceso', 'Enviada']



# Objetos (diccionarios)

En otros lenguajes de programación los objetos son comúnmente como “diccionarios”. El dato se almacena en forma de atributos y cada atributo tendrá un “key” único.

## Ejemplo:

- Objeto que almacene el tipo\_operación y la fecha\_hora\_operación para cada registro.

FECHA_HORA_OPERACION	TIPO_OPERACION
07/04/2017 15:32	Enviada
23/03/2017 12:21	Compra
23/03/2017 13:05	Aprobación de pago
27/03/2017 8:58	En proceso



OBJETO
{tipo_operación: 'Enviada', fecha_hora_operación: '07/04/2017 15:32'}
{tipo_operación: 'Compra', fecha_hora_operación: '23/03/2017 12:21'}
{tipo_operación: 'Aprobación de pago', fecha_hora_operación: '23/03/2017 13:05'}
{tipo_operación: 'En proceso', fecha_hora_operación: '27/03/2017 8:58'}

# Variant

Una variant puede almacenar cualquier tipo de dato, incluido objetos y arrays.

Ejemplo:

- Crear una lista de objetos que contengan el `tipo_operación` y la `fecha_hora_operación`. Esta lista debe estar ordenada por `fecha_hora_operación` ascendentemente.

FECHA_HORA_OPERACION	TIPO_OPERACION
07/04/2017 15:32	Enviada
23/03/2017 12:21	Compra
23/03/2017 13:05	Aprobación de pago
27/03/2017 8:58	En proceso



Evento
[ { <b>tipo_operación</b> : 'Compra', <b>fecha_hora_operación</b> : '23/03/2017 12:21'}, { <b>tipo_operación</b> : 'Aprobación de pago', <b>fecha_hora_operación</b> : '23/03/2017 13:05'}, { <b>tipo_operación</b> : 'En proceso', <b>fecha_hora_operación</b> : '27/03/2017 8:58'}, { <b>tipo_operación</b> : 'Enviada', <b>fecha_hora_operación</b> : '07/04/2017 15:32'} ]

**Nota:** Las listas y objetos pueden contener variant, por lo cual pueden contener todo tipo de dato incluido ellos mismos.



# Comandos para crear datos-semiestructurados



**TO\_ARRAY** : Cambia el tipo de dato a array

**ARRAY\_CONSTRUCT**: Convierte una o más columnas a un array.

**ARRAY\_AGG**: Funcion de ventana que pivotea los valores de entrada en una lista.

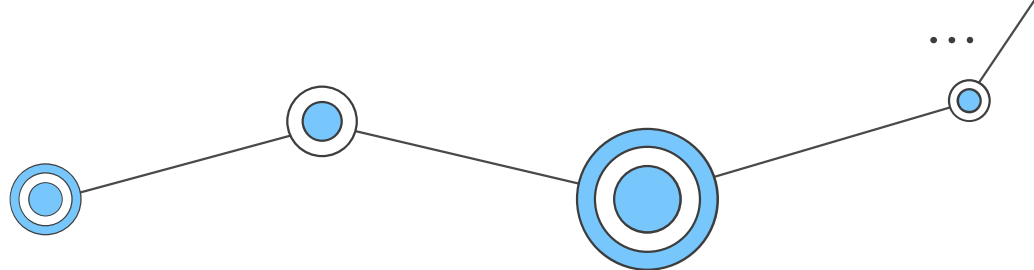
**OBJECT\_CONSTRUCT**: Convierte una o más columnas a un objeto.

Casito práctico



Para mas info sobre funciones para datos-semiestructurados:

<https://docs.snowflake.com/en/sql-reference/functions-semistructured.html>







# Acceder a elementos de una lista y objetos



Para acceder a los elementos de una lista solo necesitamos especificar la posición del elemento al que ...  
queremos acceder entre [ ]:

Ejemplo:

## Listas

Primer elemento de la lista : `Evento[0] = {tipo_operación: 'Compra', fecha_hora_operación: '23/03/2017 12:21'}`

Ultimo elemento de la lista : `Evento[array_size(evento)-1] = {tipo_operación: 'Enviada', fecha_hora_operación: '07/04/2017 15:32'}`

Hay dos formas de acceder a los elementos de un objeto:

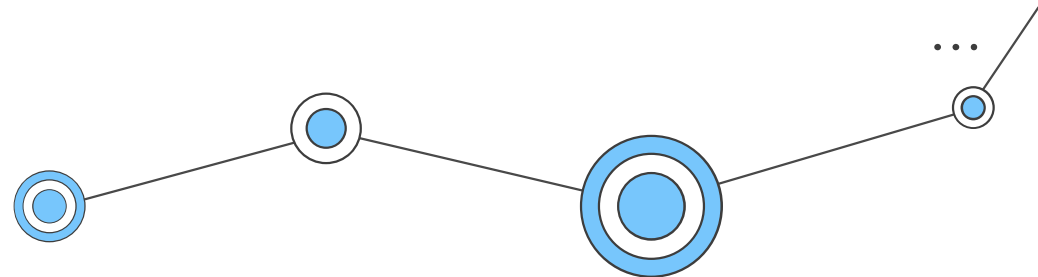
1. Especificamos el key al que queremos consultar entre comillas simples y dentro de [ ]
2. Colocamos ':' y el nombre del key

Ejemplo:

`Evento[0]['tipo_operacion'] = 'Compra'`

`Evento[0]:tipo_operación = 'Compra'`

## Objetos





# Acceder de manera “cool”




Necesidad: Obtener la información anterior y posterior a la operación *En proceso*.

```
[  
  {tipo_operación: 'Compra', fecha_hora_operación: '23/03/2017 12:21'},  
  {tipo_operación: 'Aprobación de pago', fecha_hora_operación: '23/03/2017 13:05'},  
  {tipo_operación: 'En proceso', fecha_hora_operación: '27/03/2017 8:58'},  
  {tipo_operación: 'Enviada', fecha_hora_operación: '07/04/2017 15:32'}  
]
```

## Problemática:

- Las funciones integradas por nuestro sistema de gestión de datos son limitadas. Para poder calcular este tipo de necesidades, otros lenguajes de programación poseen funciones más adaptadas.
- Debido a la gran volumetría de nuestro dato no podemos volcar toda la información a otro lugar.

Por ello, estaremos utilizando **funciones definidas por el usuario (UDF)**. Las cuales nos dan Libertad de crear funciones en diferentes lenguajes de programación sin salir del Sistema de gestión de bases de datos.



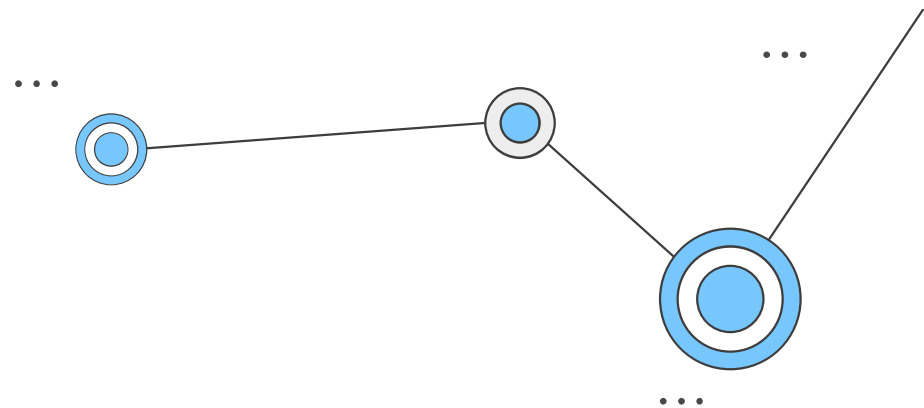
# Creación de una UDF

## Comandos

```
CREATE FUNCTION *DATABASE*.*ESQUEMA*.*NOMBRE_FUNCION*(*INPUTS: NOMBRE Y TIPO DE DATO*)
RETURNS *TIPO_DE_DATO*
LANGUAGE Python
RUNTIME_VERSION = *Version python*
HANDLER = *nombre_de_la_funcion_interna*
AS
$$
*codigo Python de nuestra funcion*
$$;
```

### Ejemplo:

```
CREATE FUNCTION GAS.WORK.existe_tipo_operacion(EVENT ARRAY,VALUE STRING)
RETURNS BOOLEAN
LANGUAGE python
RUNTIME_VERSION = '3.8'
HANDLER = 'existe_tipo_operacion'
AS
$$
def existe_tipo_operacion(event,value):
    l = any(x['tipo_operacion'] == value for x in event)
    return l
    ...
$$;
```



# Caso práctico

Ordenes ecommerce de  
tienda online

# Reporte y resultados

